The COMET Metamodel for Temporal Data Warehouses

Johann Eder[1], Christian Koncilia[1], and Tadeusz Morzy[2]

[1] Dep. of Informatics-Systems, University of Klagenfurt
{eder,koncilia}@isys.uni-klu.ac.at
[2] Institute of Computing Science, Poznan University of Technology
morzy@put.poznan.pl

Abstract. "The Times They Are A-Changing" (B. Dylan), and with them the structures, schemas, master data, etc. of data warehouses. For the correct treatment of such changes in OLAP queries the orthogonality assumption of star schemas has to be abandoned. We propose the COMET model which allows to represent not only changes of transaction data, as usual in data warehouses, but also of schema, and structure data. The COMET model can then be used as basis of OLAP tools which are aware of structural changes and permit correct query results spanning multiple periods and thus different versions of dimension data. In this paper we present the COMET metamodel in detail with all necessary integrity constraints and show how the intervals of structural stabilities can be computed for all components of a data warehouse.

1 Introduction and Motivation

A data warehouse is an integrated, materialized view over several data sources which can be conventionally structured or semi-structured data. Data Warehouses are building blocks for many information systems, in particular systems supporting decision making, controlling, revision, customer relationship management (CRM), etc.[HLV00]

Data warehouses are used for analyzing data by means of OLAP (On-Line Analytical Processing) tools which provide sophisticated features for aggregating, analyzing, and comparing data and for discovering irregularities. Data warehouses differ from traditional databases in the following aspects: They are designed and tuned for answering complex queries rather than for high throughput of a mix of updating transactions, and they typically have a longer memory, i.e., they do not only contain the actual values (snapshot data) but also historical data needed for the purposes outlined above. Historical data can be stored either directly as in temporal databases or - more frequently - as already aggregated and abstracted data.

The most popular architecture for data warehouses are multidimensional data cubes, where transaction data (called cells, fact data or measures) are described in terms of master data (also called dimension members) hierarchically organized in dimensions, where the facts of the upper levels are computed from the facts of the lower levels by some consolidation functions.

This multi-dimensional view provides long term data that can be analyzed along the time axis, whereas most OLTP (On-Line Transaction Processing) systems only supply snapshots of data at one point of time. Available OLAP systems are therefore prepared to deal with changing measures, e. g. , changing profit or turnover. Surprisingly, they are not able to deal with modifications in dimensions, e. g. , if a new branch or division is established, although time is usually explicitly represented as a dimension in data warehouses.

Consider the following example: Diagnoses for patients were represented in a data warehouse using the “International Statistical Classification of Diseases and Related Health Problems” (ICD) code. However, codes for diagnoses changed from ICD Version 9 to ICD Version 10. For instance the code for “malignant neoplasm of stomach” has changed from 151 in ICD-9 to $C16$ in ICD-10. Other diagnoses were regrouped, e.g. “transient cerebral ischaemic attacks” has moved from “Diseases of the circulatory system” to “Diseases of the nervous system”. Even worse, the same code described different diagnoses in ICD-9 and ICD-10. Other ICD-9 codes are a subset of ICD-10 codes, i.e. the granularity of codes changed (in fact ICD-10 comprises about $8,000$ unique codes, over $3,000$ more than ICD-9). The question is now: how can we get correct results for queries like “did liver cancer increase over the last 5 years”. Without knowing the above changes we will end up with incorrect results.

The reason for this disturbing property of current data warehouse technology is the implicitly underlying assumption paradigmatically visible in the Star-Schema for data warehouses that the dimensions are orthogonal. Orthogonality with respect to the dimension time means the other dimensions ought to be time-invariant. This silent assumption inhibits the proper treatment of changes in dimension data. We have to be aware that the dimensions data, i.e. the structure, the schema and the instances of the dimensions of a data warehouse may change over time.

We propose an architecture for representing the changes of a data warehouse schema and of the dimension data in a way that correct analysis of data is made possible. Since it means recognizing that the shape and content of a star (-schema) may change over time we call this a *COMET* Schema. In particular, we propose a temporal data warehouse architecture which extends multidimensional data warehouses to achieve the following features:

1. Representation of changes in master data, units and schema of data warehouses.
2. Identification of structure versions as changeless periods.
3. Provision of mappings of transaction data between structure versions.
4. Supporting queries which touch data spanning several structural versions.
5. Analysis of data according to new and old versions of the structure

In this paper we focus on the first two aspects. We propose a metamodel for data warehouses which is a temporal database of all components of a data warehouse: schema, master date (also called dimension members), hierarchical relationships etc. The data of this model is the necessary basis for achieving the other goals.

Related Work. Our concept builds on the techniques developed in temporal databases [JD98], schema evolution and schema versioning of databases [FGM00]. However, all these approaches are not designed for analytical queries like data warehouses. Therefore, extensions and adaptions for the particularities of data warehouses are necessary.

We first presented the problem and a concept for solution for simple data warehouses by transformation functions in [EK01]. The work presented here extends this approach for fact constellation schemas and provides a metamodel together with integrity constraints which covers changes in a much more detailed way, i.e., on both the schema and the instance level.

Other approaches for temporal data warehouses are [Yan01, BSH99, Vai01, CS99]. They are more (e.g, [Yan01]) or less (e.g., [CS99]) formal. To our best knowledge, only [Vai01] deals with both schema and instance modifications. However, the approach proposed in [Vai01] supports only schema/instance evolution and no versioning. Furthermore, none of the mentioned papers supports a mechanism to introduce relationships between instances in different structural versions, i.e., transformation functions for instances between different versions of structure. Hence, none of these approaches supports correct results for queries spanning multiple versions of structure.

Outline. The rest of the paper is organized as follows: In section 2 we present the concept of our temporal data warehouse approach. In section 3 we present our COMET model for temporal data warehouses. In section 4 we show how we can compute the changeless time intervals for a given dimension member. A prototype implementation of this model is sketched in section 5. Finally we draw some conclusions.

2 Temporal Multidimensional Systems

Our concept extends the well known data warehouse approach with aspects of temporal databases and schema versioning. The changes we have to cope with are not only schema changes, but also changes in the dimension data (also called master data). The dimension *Time* ensures to keep track of the history of transaction data, i.e., measures. Nevertheless, for correct query results after modifications of dimension data we have to track modifications of these data [EK01].

Therefore, we extended the well known data warehouse approach with the following aspects [EK01]:

- **Temporal extension**: dimension data has to be time stamped in order to represent their *valid time*. The *valid time* represents the time when a "fact is true in the modeled reality" [JD98].

- **Structure versions**: by providing time stamps for dimension data the need arises that our system is able to cope with different versions of structure.
- **Transformation functions**: Our system has to support functions to transform data from one structure or schema version into another.

All dimension members and all hierarchical links between these dimension members have to be time stamped with a time interval $[T_s, T_e]$ representing the valid time where T_s is the beginning of the valid time, T_e is the end of the valid time and $T_e \geq T_s$. Furthermore, we timestamp all schema definitions, i.e. dimensions, categories and their hierarchical relations, in order to keep track of all modifications of the data warehouse schema [EKM01].

If we represent all time stamps of all modifications within our data warehouse on a linear time axis the interval between two succeeding time stamps on this axis represents a structure version. This means that a structure version is a view on a temporal data warehouse valid for a given time period $[T_s, T_e]$. Therefore, within a structure version the structure of dimension data on both the schema level and on the instance level is stable. Information about structure versions can be gained from our temporal data warehouse using temporal projection and temporal selection [JD98].

The data returned by a query may originate in several (different) structure versions. Hence, it is necessary to check whether the data needed for answering the query (the relevant sub-cube) was affected by structural changes. This is important since not all structural changes affect all data. If data was affected by structural changes , it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

Using transformation functions enables us to assure that a successful analysis can be made even though there might be changes in the dimension data and dimension structure. The combination of structure versions and transformation functions enables the user to analyze data with dimension data and dimension structures “backward” or “forward” in the time axis.

3 The COMET Model

In this section we will specify our generic temporal data warehouse model COMET. The COMET model allows to register all changes of schema and structure of data warehouses.

3.1 Goals and Features

In contrast to the well known modelling techniques for data warehouses, e.g. the Star Schema modelling technique, our COMET Model allows to model data warehouses that do evolve over time.

The COMET model offers the following features for the definition of data warehouses:

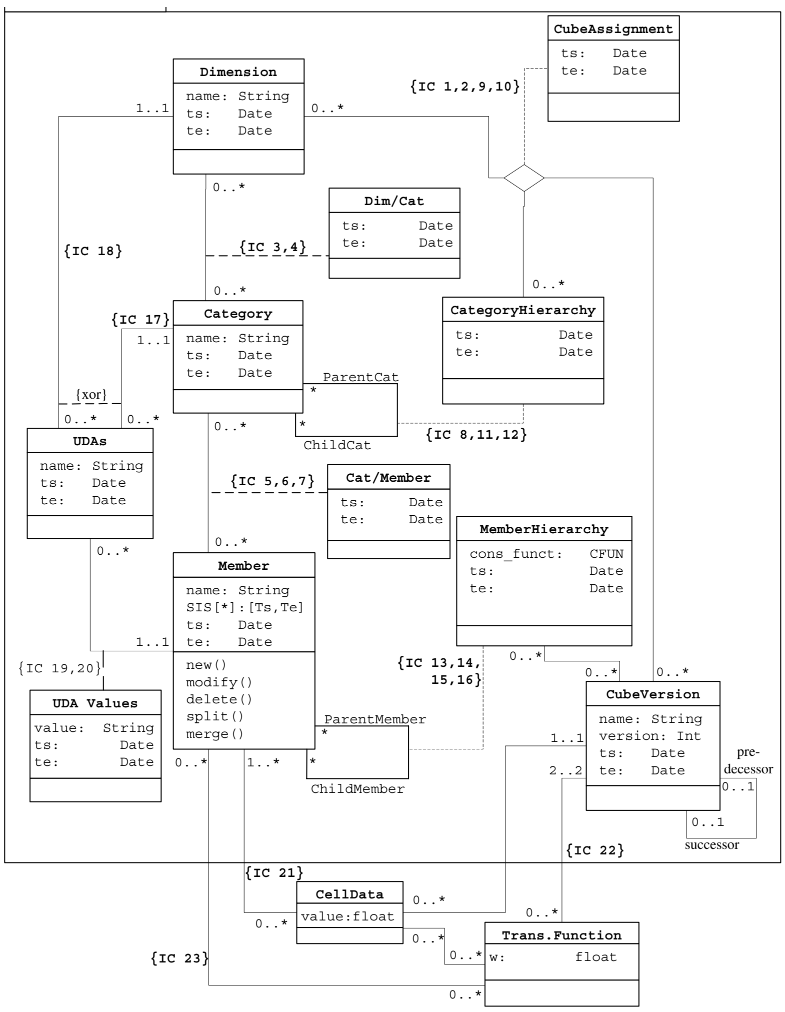

Fig. 1. The COMET Model (using UML notation)

- **Temporal Data Warehousing**: The COMET model enables us to keep track of modifications on both the instance level and the schema level.
  - Instance Level: Instances, i.e. dimension members may change over time. For example new products may become a part of the product port-folio of the company, divisions may split up into several subdivisions or the way how to compute facts may change over time.
  - Schema Level: The schema of the defined data warehouse may change over time. For example dimensions may be deleted or categories may be inserted.

- **Fact-Constellation Schemas**: Frequently it is not possible that all measures and dimensions can be captured in a single schema. Usually, a data warehouse consists of several fact tables described by several (shared or non-shared) dimensions. The COMET model allows to set up Fact-Constellation Schemas (also known as Galaxy Schemas) as proposed for example in [AV98]. Thus, it allows shared dimensions, e.g., a dimension *Products* may be part of several cubes.
- **Proportional Aggregation**: Our model supports correct aggregation even if dimension members are not disjunct. This means that one dimension member may belong to several "parents", e.g., the calendar week number 5 (which is a dimension member of the category *Weeks*) of 2002 belongs to the months January and February where four days or 4/7 belong to January and three days or 3/7 belong to February. Our model supports proportional aggregation to enable correct aggregation of non-disjunct dimension members.
- **Generic Dimensionality**: In contrast to the different OLAP models proposed so far our model fulfills E.F. Codd's sixth OLAP rule of "Generic Dimensionality" [CCS93]. He claims that each dimension must be equivalent in both its structure and operational capabilities.
  E.g., we treat the dimension *Time* that is usually a part of a data warehouse like any other dimension. Furthermore, we represent the facts of a data warehouse as a dimension *Facts*. Although there are discussions about Codd's sixth OLAP rule, it allows to apply the whole functionality of our COMET approach even to the dimension *Time* or *Facts*. For example, the dimension Time can become finer or coarser, i.e. we can insert or delete a new leaf category(say days instead of weeks). Obviously we have to register such changes and to transform values back and forth.

3.2 Elements of the COMET Model

The core of the COMET model are the classes to represent Cubes (class `CubeVersion`), Dimensions (class `Dimension`), Categories (class `Category`) and Dimension Members (class `Member`). As represented by the corresponding multiplicities in Fig. 1, a cube consists of several dimensions and each dimension consists of several categories.

Categories are represented in the class `Category` and are in a hierarchical order. E.g., the categories "Country", "State", "Region" and "City" are in the following order $Country \leftarrow State$, $Country \leftarrow Region$, $State \leftarrow City$ and $Region \leftarrow City$ were $X \leftarrow Y$ means Y rolls-up to X. This is represented by the recursive association of the class `Category`.

The same applies for dimension members which are also in a hierarchial order represented by the recursive association of the class `Member`. For example the dimension members "USA", "Texas", "South" and "Dallas" are in the order $USA \leftarrow Texas$, $USA \leftarrow South$, $Texas \leftarrow Dallas$ and $South \leftarrow Texas$.

As both hierarchical relations between categories and between dimension members may change over time we have to represent the valid time of these relations with timestamp represented in the classes `CategoryHierarchy` and `MemberHierarchy`.

Furthermore, we have to represent how data may be aggregated. We represent this with the attribute `cons_func` in the class `MemberHierarchy`. Moreover, this attributes allows the representation of proportional aggregation functions for non-disjunct members. For example, an salesman may be assigned to two divisions $Div1$ and $Div2$. The amount of sales attained by this salesman may be assigned with the factors 40% and 60% to the divisions $Div1$ and $Div2$.

Another important aspect that the COMET model has to fulfill is that the relation between dimensions and categories may change over time. The same applies for the relation between categories and dimension members. This is represented with the time stamps (attributes $\mathtt{t}_s$ and $\mathtt{t}_e$) in the association classes `Dim/Cat` and `Cat/Member`.

We have to deal with different versions of a cube due to the fact that the schema and/or instances of a cube may change over time and the fact that the COMET model supports not only schema evolution but schema versioning. Each version of a cube may have a preceding and a succeeding version and each version is valid for a given valid time. We represent this with the the class `CubeVersion` and the recursive relation for this class.

The COMET model supports transformation of data from one structure and/or schema version into the (immediate) succeeding or preceding version. We represent these transformation functions within the class `Trans.Function`. Each transformation function transforms several cell data entries from exactly one version into its preceding or succeeding version. Moreover, several transformation functions may be defined to transform a cell value from one version into its preceding and succeeding version.

Transformation functions are only allowed between contiguous cube version (two versions V_i and V_k are contiguous if $T_{s,i} = T_{e,k} + Q$ or if $T_{s,k} = T_{e,i} + Q$ where Q is the defined chronon of the data warehouse). As described in section 2 we can represent these transformation functions as matrices. In order to increase query performance, we are able to automatically compute transformation matrices between two non-contiguous version by multiplying all corresponding transformation matrices. Consider for example two transformation matrices $M_{1\to2}$ to transform data from version V_1 into version V_2 and $M_{2\to3}$ to transform data from version V_2 into version V_3. By multiplying these transformation matrices we can compute a transformation function $M_{1\to3}$ to transform data directly from version V_1 into V_3.

As described above the COMET model allows us to assign different dimensions to a cube. On the other hand each dimension may be assigned to several cubes in order to allow fact-constellation schemas. Furthermore, if a dimension is assigned to more then one cube it may consist of a different structure, i.e. a different set of categories and their hierarchical assignments, in each of these cubes. This is represented by the tertiary association between the classes `CubeVersion`, `Dimension` and `CategoryHierarchy`.

Furthermore, each dimension member may have different *User Defined Attributes* (UDA), e.g. the products stored in the data warehouse may have a "Color" and a "Weight". Hence, UDAs are attributes describing a dimension member. In contrast to dimensions UDAs do not allow the appliance of OLAP functionality, e.g. Drill-Down, Roll-Up, Slice, etc.

COMET allows the definition of UDAs for each defined `Dimension` or `Category`. The UDAs defined for a certain dimension define the UDAs applicable for all dimension members assigned to this dimension (all dimension members that are assigned to a category which is assigned to the specific dimension). The UDAs defined for a certain category define the UDAs applicable for all dimension members assigned to this category. E.g., one may want to assign a "Color" to each product and hence define this UDA through the dimension *Products*, but only products that do belong to the category "Video Cassette Recorder" have a UDA "Number of Heads". The attributes $\mathtt{t}_s$ and $\mathtt{t}_e$ in the class `UDAs` represent the valid time of a UDA.

The association class `UDAValues` specifies the value of a specific UDA for a specific dimension member. The attributes $\mathtt{t}_s$ and $\mathtt{t}_e$ in the class `UDA Values` represent the valid time of a value defined for a specific UDA and a specific dimension member.

The measures of the defined data warehouse are stored in the class `CellData`. A measure, i.e., a cell in a n-dimensional data cube contains a value and is referenced by a vector $\nu = (DM_{D_1}, ..., DM_{D_N})$ where DM_{D_i} is a dimension member DM that is assigned to a dimension D_i [Kur99].

Due to the temporal extensions that are a part of the COMET model a lot of integrity constraints have to be taken into consideration. In the next section, we will discuss these constraints in detail.

3.3 Integrity Constraints

We will now discuss the constraints that have to be fulfilled in order to guaranty the integrity of our temporal data warehouse model.

A basic constraint is that for all time stamps $[T_s, T_e]$ in the COMET model $T_s \leq T_e$ has to be true, i.e., a *temporal component* may not end before it starts. A *temporal component* is an object of any class that has attributes to represent the valid time (attributes $\mathtt{t}_s$ and $\mathtt{t}_e$), e.g., a dimension, a category, a dimension-member and so on.

In order to give a formal description of the integrity constraints we introduce the predicates *overlaps* and *exists_in* as follows:

- *overlaps*(C_i, C_j): describes that the valid time of temporal component C_i overlaps the valid time of temporal component C_j and vice-versa.
  overlaps(C_i, C_j) is true if $\exists t \bullet (T_s^{C_i} \leq t \leq T_e^{C_i}) \wedge (T_s^{C_j} \leq t \leq T_e^{C_j})$, i.e. it is fulfilled, if there exists at least one point in time where both temporal components are valid.

- $exists\_in(C_i, C_j)$: describes that the time interval representing the valid time of temporal component C_i is a subset of the time interval representing the valid time of temporal component C_j.
  $exists\_in(C_i, C_j)$ is true if $\forall t \in [T_s^{C_i}, T_e^{C_i}] \bullet t \in [T_s^{C_j}, T_e^{C_j}]$.

Furthermore, we will use the notation $A.X$ as a reference to the component X of the assignment A. An assignment may be an object of the classes `Dim/Cat`, `Cat/Member`, `CubeAssignment`, `CategoryHierarchy` or `MemberHierarchy`.

• **IC 1:** A dimension D can be assigned to a Cube C if the valid time intervals of both components are overlapping. Furthermore, both components have to be valid within each timepoint of the valid time of the assignment:

$$\forall A \in \text{CubeAssignment} : exists\_in(A, D) \wedge exists\_in(A, C) \tag{1}$$

• **IC 2:** Within the valid time interval of an assignment $A^C.D$ between a dimension D and a cube C the dimension D must not be assigned more than once to this cube:

$$\forall D_x, D_y \in \{A^C.D_1, ...A^C.D_N\} : D_x = D_y \rightarrow \neg overlaps(A^C.D_x, A^C.D_y) \tag{2}$$

• **IC 3:** A category G can be assigned to a dimension D if the valid time intervals of both components are overlapping. Furthermore, both components have to be valid within each timepoint of the valid time of the assignment:

$$\forall A \in \text{Dim/Cat} : exists\_in(A, G) \wedge exists\_in(A, D) \tag{3}$$

• **IC 4:** Within the valid time interval of an assignment $A^D.G$ between a category G and a dimension D the category G must not be assigned more than once to this dimension:

$$\forall G_x, G_y \in \{A^D.G_1, ...A^D.G_N\} : G_x = G_y \rightarrow \neg overlaps(A^D.G_x, A^D.G_y) \tag{4}$$

• **IC 5:** A dimension member M can be assigned to a category G if the valid time intervals of both components are overlapping. Furthermore, both components have to be valid within each timepoint of the valid time of the assignment:

$$\forall A \in \text{Cat/Member} : exists\_in(A, M) \wedge exists\_in(A, G) \tag{5}$$

• **IC 6:** Within the valid time interval of an assignment $A^G.M$ between a dimension member M and a category G the dimension member M must not be assigned more than once to this category:

$$\begin{gathered}\forall\ M_x, M_y \in \{A^G.M_1, ...A^G.M_N\} : \\ M_x = M_y \rightarrow \neg overlaps(A^G.M_x, A^G.M_y)\end{gathered} \tag{6}$$

• **IC 7:** Furthermore, within the valid time interval of an assignment A between a dimension member M and a category G - were G is again assigned to a dimension D_1 - the dimension member M must not be assigned to another category assigned to a different dimension D_2 were D_1 and D_2 are assigned to the same cube C:

$$\begin{aligned}
&\forall\,(M,G_i),(M,G_j) \in \text{Cat/Member} \\
&\quad \forall t : T_s^{(M,G_i)} \le t \le T_e^{(M,G_i)} \wedge T_s^{(M,G_j)} \le t \le T_e^{(M,G_j)} \bullet \\
&\quad G_i \ne G_j \wedge \\
&\quad (((G_j,D) \in \text{Dim/Cat} \wedge (G_i,D) \in \text{Dim/Cat}) \vee \\
&\quad ((G_j,D_1) \in \text{Dim/Cat} \wedge (G_i,D_2) \in \text{Dim/Cat} \wedge \\
&\quad (D_1,C_1) \in \text{CubeAssignment} \wedge (D_2,C_2) \in \text{CubeAssignment} \wedge \\
&\quad C_1 \ne C_2))
\end{aligned} \tag{7}$$

Please note that this constraint does allow a dimension member to be assigned to different categories of the same dimension, e.g. “Diet-Cola” may be part of “Diet-Drinks” and “Soft-Drinks” where both “Diet-Drinks” and “Soft-Drinks” are part of dimension “Products”.

• **IC 8:** The assignment between two categories G_1 and G_2 as *ChildCat* and *ParentCat* within the class `CategoryHierarchy` is allowed if the valid time intervals of both categories are overlapping. Furthermore, both categories have to be valid within each timepoint of the valid time of the assignment:

$$\forall A \in \text{CategoryHierarchy} : \mathit{exists\_in}(A,G_1) \wedge \mathit{exists\_in}(A,G_2) \tag{8}$$

• **IC 9:** An assignment between a dimension D and two categories G_1 and G_2 (`CategoryHierarchy`) may be a part of the relation `CubeAssignment` if both categories are assigned to the dimension D:

$$\begin{aligned}
&\forall\, A \in \text{CubeAssignment} : \exists y_1, y_2 \in \text{Dim/Cat} \bullet \\
&\quad y_1.D = A.D \wedge y_2.D = A.D \wedge \\
&\quad (y_1.G, y_2.G) = A.(G_1,G_2)
\end{aligned} \tag{9}$$

• **IC 10:** Furthermore, all components (a dimension D, an assignment AG between two categories and a `CubeVersion` C) that are a part of a relation `CubeAssignment` have to exist during the valid time of the relation:

$$\begin{aligned}
&\forall\, A \in \text{CubeAssignment} : \mathit{exists\_in}(A.D,A) \wedge \mathit{exists\_in}(A.AG,A) \wedge \\
&\quad \mathit{exists\_in}(A.C,A)
\end{aligned} \tag{10}$$

• **IC 11:** The assignment A between two categories as *ChildCat* and *ParentCat* is allowed if there does not already exist an assignment for the time interval of the assignment A:

$$\forall A_i, A_j \in \text{CategoryHierarchy} : (A_i.G, D) = (A_j.G, D) \rightarrow A_i = A_j \tag{11}$$

• **IC 12:** There must not be any cycles within the assignments in the class `CategoryHierarchy`:

$$childsC(P) = \{x \in \text{Category} : \exists A \in \text{CategoryHierarchy} \bullet A = (P, x) \cup childsC(x)\} \tag{12}$$

$$\nexists A \in \text{CategoryHierarchy} : (A.P, A.x) \wedge P \in childsC(P) \tag{13}$$

• **IC 13:** The assignment between two dimension members as *ChildMember* and *ParentMember* within the class `MemberHierarchy` is allowed if both members M_x and M_y are assigned to the same category G for each timepoint t within the time interval representing the valid time of the assignment. Furthermore, both components (*ChildMember* and *ParentMember*) have to be valid within the time interval of the assignment:

$$\begin{aligned} &\forall\ A \in \text{MemberHierarchy} : (A.M_x, G) \in \text{Cat/Member} \wedge \\ &\quad G \in \text{Category} \wedge \\ &\quad (A.M_y, G) \in \text{Cat/Member} \wedge \\ &\quad exists\_in(A, (A.M_x, G)) \wedge \\ &\quad exists\_in(A, (A.M_y, G)) \end{aligned} \tag{14}$$

• **IC 14:** The assignment A between two dimension members as *ChildMember* and *ParentMember* is allowed if there does not already exist an assignment for the time interval of the assignment A:

$$\forall A_i, A_j \in \text{MemberHierarchy} : (A_i.M, G) = (A_j.M, G) \rightarrow A_i = A_j \tag{15}$$

• **IC 15:** An assignment A between two dimension members M_1 and M_2 as `MemberHierarchy` is allowed if there exists an assignment between two categories G_1 and G_2 as `CategoryHierarchy` and if both components of the `MemberHierarchy` M_1 and M_2 are assigned to both components of the `Category-Hierarchy` G_1 and G_2 such that M_1 is assigned to G_1 and M_2 is assigned to G_2:

$$\begin{aligned} &\forall\ A \in \text{MemberHierarchy} : \\ &\quad \exists A_{CH} \in \text{CategoryHierarchy} \wedge \exists B_1, B_2 \in \text{Cat/Member} \bullet \\ &\quad A.M_1 = B_1.M \wedge A.M_2 = B_2.M \wedge \\ &\quad A_{CH}.G_1 = B_1.G \wedge A_{CH}.G_2 = B_2.G \end{aligned} \tag{16}$$

• **IC 16:** There must not be any cycles within the assignments in the class `MemberHierarchy`:

$$childsM(P) = \{x \in \text{Member} : \exists A \in \text{MemberHierarchy} \bullet A = (P, x) \cup childsM(x)\} \tag{17}$$

$$\nexists A \in \text{MemberHierarchy} : (A.P, A.x) \wedge P \in childsM(P) \tag{18}$$

• **IC 17:** An assignment between an UDA U and a category G is allowed, if the valid time of the UDA is within the valid time of the category:

$$\forall G \in \text{Category} : exists\_in(G.UDA, G) \tag{19}$$

• **IC 18:** An assignment between an UDA U and a dimension D is allowed, if the valid time of the UDA is within the valid time of the dimension:

$$\forall D \in \text{Dimension} : exists\_in(D.UDA, D) \tag{20}$$

• **IC 19:** An assignment between an UDA U and a dimension member M within `UDA Values` is allowed, if the valid time of the UDA value is within the valid time of both components:

$$\forall A \in \text{UDAValues} : exists\_in(A, M) \wedge exists\_in(A, U) \tag{21}$$

• **IC 20:** Furthermore, as assignment between an UDA U and a dimension member M within `UDA Values` is allowed, if the UDA is assigned to a category G and the dimension member is assigned to this category, or if the UDA is assigned to a dimension D and the dimension member is assigned to this dimension:

$$\begin{aligned} &\forall x \in \text{UDAValues} : \\ &\quad (\exists y \in \text{Cat/Member} \bullet \\ &\qquad y.M = x.M \wedge y.G = x.U.G) \vee \\ &\quad (\exists y \in \text{Dim/Cat} \bullet \\ &\qquad \exists z \in \text{Cat/Member} \bullet \\ &\qquad\quad z.G = y.G \wedge z.M = x.M \wedge y.D = x.U.D) \end{aligned} \tag{22}$$

• **IC 21:** The number of dimension members used to reference a `CellData` must be equal to the number of dimensions assigned to the cube: Let $\mathbb{M} = \{M_1, ..., M_n\}$ be the set of dimension members used to reference a cell V and let $\mathbb{D} = \{D_1, ..., D_m\}$ be the set of dimensions assigned to a cube C. V may be assigned to C if $n = m$.

• **IC 22:** A transformation function `Trans.Function` may be used to transform cell values between two contiguous versions of a cube CV_1 and CV_2 (Q is the defined chronon of the data warehouse):

$$\begin{aligned} \forall x \in \text{Trans.Function} \bullet\ & (x.CV_1.T_e + Q = x.CV_2.T_s) \vee \\ & (x.CV_1.T_s - Q = x.CV_2.T_e) \end{aligned} \tag{23}$$

Transformation functions between non-contiguous version can be automatically computed by multiplying the given transformation matrices as proposed in [EK01].

• **IC 23:** A transformation function `Trans.Function` may be applied to lower-level dimension members only (dimension members without successors):

$$\begin{aligned} &\forall x \in \text{Trans.Function} : \\ &\quad \nexists A \in \text{MemberHierarchy} \bullet A(x.Member, \_\_) \end{aligned} \tag{24}$$

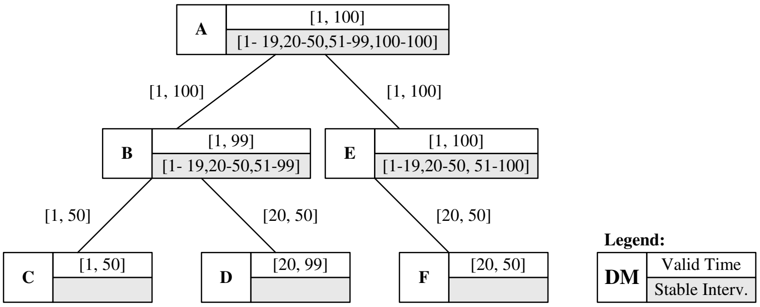

Fig. 2. Stable intervals sets

4 Stable Intervals

In this section, we will discuss an important aspect of our COMET approach – called *Stable Intervals Sets* – that enables us to increase query performance of queries spanning multiple structure versions.

The basic idea is that we want to compute the largest stable interval for upper-level dimension members (a dimension member with at least one successor). A stable interval of an upper-level dimension member DM is an interval in which no modification to another component does affect query results for DM.

As described in section 2 each dimension member and each hierarchical relation between dimension members has a timestamp that represents the valid time of the corresponding object. Nevertheless, this does not mean that changes "beneath" an upper-level dimension member do not affect the data computed for this dimension member during the given time interval of the valid time.

Consider for example a temporal data warehouse that stores information about a faculty. The faculty did not change but the departments within the faculty, the staff, etc. might have changed. Hence, if a query retrieves data for the faculty within the valid time interval of the faculty it still might be affected by wrong computations due to structural evolution of departments, etc. On the other hand, we do not want to perform transformations for all queries, with several structure versions within the relevant time interval of the query, if these structure versions stem from changes in parts from the data warehouse which are not relevant for the given query.

The stable intervals set of a dimension member is now a set of time intervals where for each interval the structures influencing (derived) cell-data for this dimension member did not change. So if a query stays with a stable interval, there is no need to transform data.

More formal we can define the following: let $G_{DM} = (\mathcal{N}, \mathcal{E})$ be a directed graph that represents all dimension members their hierarchical relations between dimension members consolidating in a dimension member DM where $\mathcal{N}$ is the set of all dimension members (=nodes) and $\mathcal{E}$ is the set of all hierarchial relations

```
REM: O.sis =  stable intervals set for object O
REM: DM = Dimension Member that changed
REM: T = Timestamp of Dimension Member DM
REM: ℙ = Set of all parents of DM

ComputeSIS(↓T:TimeStamp, ↓DM: Member)
{
   $\mathbb{P} = \{x \in \texttt{Members} : \exists y \in \texttt{MemberHierarchy} \bullet y(x, \texttt{DM})\}$
   FOR ALL $p \in \mathbb{P}$ :
      IF $\nexists y \in p.sis \bullet y.t_s = T.t_s \vee y.t_e = T.t_s$
            I = $y \in p.sis \bullet y.t_s < T.t_s \wedge y.t_e > T.t_s$
            $p.sis = p.sis \cup [T.t_s, I.t_e]$
            I.$t_e$ =T.$t_s - 1$
      ENDIF
      IF $\nexists y \in p.sis \bullet y.t_s = T.t_e \vee y.t_e = T.t_e$
            I = $y \in p.sis \bullet y.t_s < T.t_e \wedge y.t_e > T.t_e$
            $p.sis = p.sis \cup [T.t_e, I.t_e]$
            I.$t_e$ =T.$t_e - 1$
      ENDIF
      ComputeSIS(T, p)
   ENDFOR
}
```

Fig. 3. The `ComputeSIS` algorithm

between those dimension members (=edges). Both $\mathcal{N}$ and $\mathcal{E}$ are defined through the class `MemberHierarchy` of our COMET model.

The Stable Intervals Set for leaf dimension members (a dimension member without successors) is equivalent to the valid time defined for this member. The Stable Intervals Set of a non-leaf dimension member DM changes with each modification within the graph G_{DM} as defined below. In the COMET model the Stable Intervals Set is represented with the attribute `SIS` of the class `Member`.

Figure 2 shows how we represent the set of stable intervals within dimension members. For instance, the set of stable intervals of the dimension member B is derived from the valid time of all succeeding dimension members (C and D) and hierarchical relations ($B \leftarrow C$ and $B \leftarrow D$).

Figure 3 sketches the algorithm to propagate changes of dimension members that affect the Stable Intervals Set. The algorithm to propagate changes of hierarchical relations ($Parent \leftarrow Child$) is similar to the proposed algorithm except that it first computes the Stable Intervals Set for the $Parent$ and then calls `ComputeSIS`(T, P) where T is the timestamp of the hierarchical relation and P is the $Parent$.

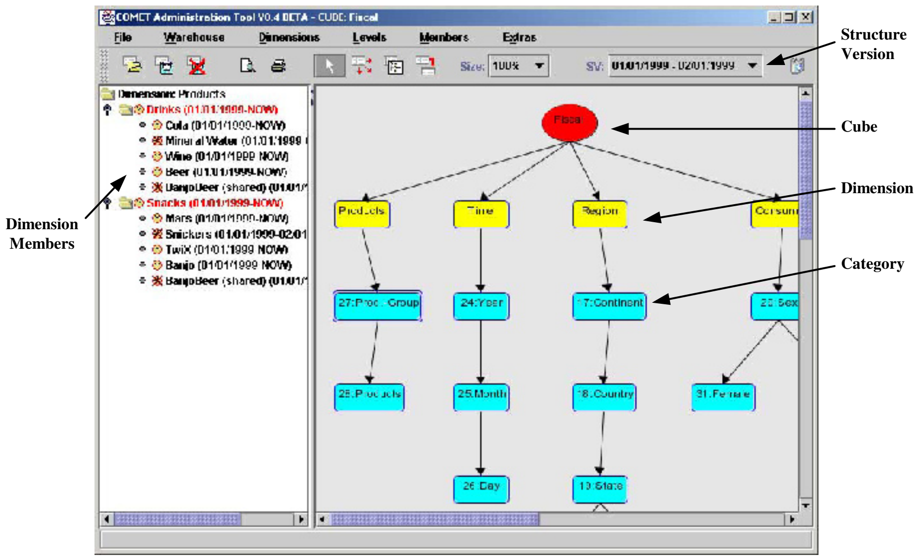

Fig. 4. The COMET administration tool

5 Implementation

In [EKM01] we discussed different implementation architectures. We are currently prototypically implementing a temporal data warehouse based on our COMET model with the *indirect approach* [EKM01].

The COMET prototype consists of three parts: the Temporal Data Warehouse (holds the required information about structure versions, cell data and transformation functions) implemented in Oracle 8.1, the Transformer (maps all required cell values from all required structure versions into the chosen base structure version by using the defined transformation functions) and the Administration Tool (allows to define the schema and structure of a temporal data warehouse, to modify schema and structure and to import cell data into the temporal data warehouse).

In Fig. 4 we show a screen shot of the main window of the Administration Tool. This window shows a cube and all assigned components (dimensions, categories and dimension members). The administrator may select a specific version of this cube by selecting the corresponding structure version in the selection list SV (top right).

The main idea of the indirect approach is that the Transformer generates a data mart for each structure version needed by the user. In most cases, this will only be the actual structure version. Each data mart consists of all fact data that are valid for the same time interval as the corresponding structure version plus it consists of all fact data that could be transformed by the defined transformation functions from all other structure versions.

Therefore, the user defines his/her base structure version by selecting a specific data mart. This base structure version determines which structure has to be used for the analysis. In most cases this will be the current structure version. However, in some cases, e.g. for auditing purposes, it will be of interest to use an "older" structure version.

We are implementing this approach with Oracle 8.1i as basis for our Temporal Data Warehouse and Hyperion Essbase (Release 6) as front end that holds our data marts. As we use a standard OLAP database for each data mart, the main advantage of the indirect approach is that each data mart offers the whole OLAP functionality, e.g., drill-down, roll-up, slice, dice, etc.

6 Conclusions

Unfortunately, many of our information systems are ill prepared for change and, surprisingly, multidimensional data warehouse systems are among those. Naturally, it is vital for the correctness of results of OLAP queries that modifications of dimension data is correctly taken into account. E.g., when the economic figures of European countries over the last 20 years are compared on a country level, it is essential to be aware of the re-unification of Germany, the separation of Czechoslovakia, etc.

Business structures and even structures in public administration are nowadays subject to highly direct changes. Comparisons of data over several periods, computation of trends, computation of benchmark values from data of previous periods have the necessity to correctly and adequately treat changes in dimension data. Otherwise we face meaningless figures and wrong conclusions triggering bad decisions. From our experience we could cite too many such cases.

The COMET model we propose in this paper allows to register all changes of schema and structure of data warehouses. The innovation lies in the complete registration and temporal attribution of *all* elements of a data warehouse. This is then the basis for OLAP tools, transformation operations, the derivation of (correctly) star shaped data marts etc. with the goal to reduce incorrect OLAP results.

References

[AV98] C. Adamson and M. Venerable. *Data Warehousing Design Solutions.* Wiley, New York, 1 edition, 1998. 88

[BSH99] M. Blaschka, C. Sapia, and G. Höfling. On Schema Evolution in Multidimensional Databases. In *Proc. of the DaWak'99 Conference*, 1999. 85

[CCS93] E. Codd, S. Codd, and C. Smalley. *Providing OLAP to User-Analysts: An IT Mandate.* Hyperion Solutions Corporation, California, 1993. 88

[CS99] P. Chamoni and S. Stock. Temporal Structures in Data Warehousing. In *Data Warehousing and Knowledge Discovery (DaWaK) 1999*, pages 353–358, Italy, 1999. 85

[EJS98] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise.* Number LNCS 1399. Springer-Verlag, 1998.

[EK01] J. Eder and C. Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proc. of the DaWak 2001 Conference*, 2001. 85, 94

[EKM01] J. Eder, C. Koncilia, and T. Morzy. A Model for a Temporal Data Warehouse. In *Proc. of the Int. OESSEO 2001 Conference*, Rome, Italy, 2001. 86, 97

[FGM00] E. Franconi, F. Grandi, and F. Mandreoli. Schema Evolution and Versioning: a Logical and Computational Characterisation. In *Workshop on Foundations of Models and Languages for Data and Objects*, 2000. 85

[HLV00] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual Data Warehouse Design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW 2000)*, Stockholm, 2000. 83

[JD98] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*, pages 367–405. Springer-Verlag, 1998. in [EJS98]. 85, 86

[Kur99] A. Kurz. *Data Warehousing - Enabling Technology.* MITP-Verlag, 1999. 90

[Vai01] A. Vaisman. *Updates, View Maintenance and Time Management in Multidimensional Databases.* Universidad de Buenos Aires, 2001. Ph.D. Thesis. 85

[Yan01] J. Yang. *Temporal Data Warehousing.* Stanford University, 2001. Ph.D. Thesis. 85